REMARKS

This is intended as a full and complete response to the Final Office Action dated June 3, 2005, having a shortened statutory period for response set to expire on September 3, 2005. Applicants submit this response to place the application in condition for allowance or in better form for appeal. Please reconsider the claims pending in the application for reasons discussed below.

Claims 1-29 are pending in the application. Claims 1-29 remain pending following entry of this response. Claim 1 has been amended. Applicants submit that the amendments do not introduce new matter.

Claim Rejections - 35 U.S.C. § 103

Claims 1-29 are rejected under 35 U.S.C. § 103(a) as being unpatentable over Borland's, "Turbo C++, User's Guide" (hereinafter *Borland*) in view of *Kesselman et al.* (US 6,678,884, hereinafter *Kesselman*). Applicants respectfully traverse the rejection.

The Examiner bears the initial burden of establishing a *prima facie* case of obviousness. *See* MPEP § 2142. To establish a *prima facie* case of obviousness three basic criteria must be met. First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one ordinary skill in the art, to modify the reference or to combine the reference teachings. Second, there must be a reasonable expectation of success. Third, the prior art reference (or references when combined) must teach or suggest all the claim limitations. *See* MPEP § 2143. The present rejection fails to establish at least the third criteria.

Claims 1, 14, and 27 recite the limitation of executing at a first task when a run command is received, wherein: (a) the first task comprises determining a first kill variables set ... and claims 1 and 14 further recite executing a second task when a set step command for a statement is received, wherein (b) the second task comprises determining a second kill variables set ....

In support of the rejection, the Examiner cites material from a User's Guide for the Borland IDE that provides a debugger program (Turbo Debug). The cited portions

include sections teaching a user how to interact with the Turbo Debug product. For example, at pages 237-241, the cited material describes how to set a "watch" for a variable of a program being debugged. Using the Borland product, a watched variable is monitored by the debugging system and displayed in a window that shows a user the value of a selected variable during program debugging (e.g., during execution up to a breakpoint or executing individual lines of code using a step function provided by the debugger). Nothing in the material cited by the examiner discloses executing a task to determine a first kill variable set, a second kill variable set, or for that matter ***any*** kill variable set.

Rather, as one would expect, the Borland User's Guide teaches a user how to interact with Borland's commercial software product, and ***not*** how the *Borland* product performs internally. That is, the Borland User's Guide informs a user of the features provided by Turbo Debug, but does not teach or suggest how to implement the features of a debugger, including the feature allowing users to monitor "changing values automatically by setting watches," *Borland,* p. 237, or how to improve upon the efficiency of such features. Although disclosing a window to display watched variables, *Borland* fails to disclose *any* information regarding how the Turbo Debug product retrieves and displays the value for a given variable in general, or discloses using a executing a task to determine a kill variables set in particular.

Moreover, the Examiner concedes that the Borland User's Guide fails "to explicitly state first kill variables set comprising only those variables which may be affected by the execution of a program from a particular point ... to a breakpoint and second kill variables set comprising only those variables which may be affected by execution of the statement." *See Final Office Action,* p. 3.

In fact, as previously noted, the Borland User's Guide fails to disclose a kill variables set at all. As the Borland User's Guide does not disclose this element, the Examiner turns to *Kesselman* and asserts that *Kesselman* "demonstrated that it was known at the time of invention to utilize code point reaching variable sets (column 6, line 59 to column 7, line 12)." *See Final Office Action,* p. 3. Respectfully, the "reaching

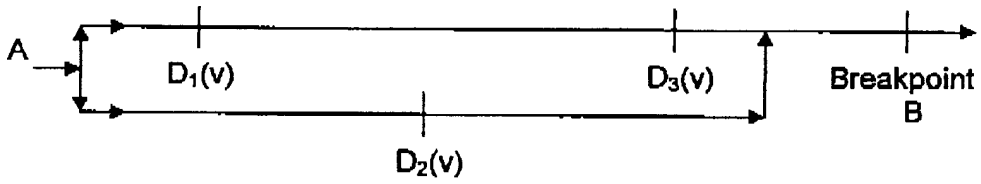
definitions" disclosed in *Kesselman* fails to teach or suggest a kill variables set in the manner claimed by Applicants.

Keeselman provides a method for determining the correct value of a variable to display during debugging session when debugging optimized code. This is required as the optimized code may change variables in an order different than an expected value determined from the source code. In describing this method *Kesselman* provides a couple of relevant definitions:

> A definition of a variable V is a statement in source program in which a value is assigned to variable V. The definition D of a variable V reaches breakpoint B along the path P ending at breakpoint B if and only if D is the last statement of P that is a definition of V.
>
> A reaching definition set of the variable V at a breakpoint V is a set of all V's definitions that may reach B along some simple (without repeated nodes) path terminating at B.

Kesselman, 3:25-34. The following diagram illustrates these definitions:

Source Code Path of Execution of Program P



As can be seen from this illustration, the variable v includes three definitions "D." Each definition represents a statement in a program assigning a value to the variable v. For example, assuming v is an integer variable, the assignments might be source code statements like:

```
v = a+2;
v=5;
v++;
```

Further, in this illustration, the path of execution begins at "A" and proceeds along one of two paths, eventually reaching breakpoint B. Thus, the "reaching definition set of v", as defined in *Kesselman*, includes D_2, and D_3, as each definition may be reached along

a path that reaches breakpoint B. D_1, however, is not in the "reaching definition" of the variable v as it is not "the last statement of P" that assigns a value to v.

The material cited by the Examiner from *Kesselman* uses these definitions for determining the correct value for a source code variable V to display when debugging optimized code. For example, a portion of the material cited by the Examiner provides: "The calculation of source reaching set of a variable involves checking all possible source flow paths that end at the breakpoint. The source reaching set comprise from [sic] all the definitions of the variable that may reach the breakpoint along one of these paths." *Kesselman*, 7:8-12. Thus, the source reaching set of v includes D_2 and D_3. The reaching set is then used to determine the "expected value of a variable" at breakpoint B. That is, "the value that would be predicted by examining the source code and knowing the relevant context." *Kesselman*, 3:40-43.

Determining a reaching set of "definitions" (as used in *Kesselman*) for a variable, however, fails to teach or suggest executing a task that determines a kill variables set that comprises only those variables being monitored by the debugging environment whose values may become out of sync with the corresponding values of those monitored variables, while being used by a program being debugged. Rather as stated, the "source reaching set comprise from all the definitions of the variable that may reach the breakpoint along one of these paths." *Kesselman*, 7:8-12. Calculating all the points of execution that *may* change a variable is not the same a debugger configured to determine a kill variables set that will determine which variables will be modified by execution of the program, and updating a value for these variables maintained by the debugger for only these variables.

Applicants' specification further highlights the distinction in describing the kill variables set: "The information about which variables may be affected can be stored as individual variables or sets of variables. Variables whose value may be changed are referred to herein as 'killed variables' or 'kill variables.'" *Specification*, ¶ 30. While the Examiner should not import limitations from the specification into the claims, the Examiner is obligated to read the claims in light of the specification. And doing so in

this case clearly distinguishes the kill variables set of claims 1, 14 and 27 from the "reaching definition set for a variable" disclosed in *Kesselman*.

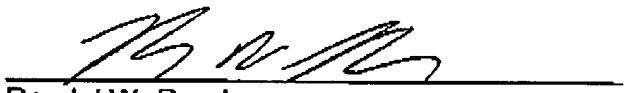
Accordingly, Applicants assert that *Borland* in view of *Kesselman* fails to disclose a kill variables set in the manner claimed. Therefore, for all the reasons stated above, Claims 1, 14, 27, along with the claims dependent therefrom, are believed to be allowable, and allowance of these claims is, therefore, respectfully requested.

Conclusion

Having addressed all issues set out in the office action, Applicants respectfully submit that the claims are in condition for allowance and respectfully request that the claims be allowed.

If the Examiner believes any issues remain that prevent this application from going to issue, the Examiner is strongly encouraged to contact the undersigned attorney to discuss strategies for moving prosecution forward.

Respectfully submitted,



Randol W. Read
Registration No. 43,876
MOSER, PATTERSON & SHERIDAN, L.L.P.
3040 Post Oak Blvd. Suite 1500
Houston, TX 77056
Telephone: (713) 623-4844
Facsimile: (713) 623-4846
Attorney for Applicants